

# 1 Setting

## 1.1 vimrc

```
1 set ts=4 sts=4 sw=4
2 set ai si nu
```

# 2 Math

## 2.1 basic arithmetic

```
1 typedef long long ll;
2 typedef unsigned long long ull;
3
4 // calculate ceil(a/b)
5 // |a|, |b| <= (2^63)-1 (does not cover -2^63)
6 ll ceildiv(ll a, ll b) {
7     if (b < 0) return ceildiv(-a, -b);
8     if (a < 0) return (-a) / b;
9     return ((ull)a + (ull)b - 1ull) / b;
10 }
11
12 // calculate floor(a/b)
13 // |a|, |b| <= (2^63)-1 (does not cover -2^63)
14 ll floordiv(ll a, ll b) {
15     if (b < 0) return floordiv(-a, -b);
16     if (a >= 0) return a / b;
17     return -(ll)(((ull)(-a) + b - 1) / b);
18 }
19
20 // calculate n^k % m
21 ll modpow(ll n, ll k, ll m) {
22     ll ret = 1;
23     n %= m;
24     while (k) {
25         if (k & 1) ret = ret * n % m;
26         n = n * n % m;
27         k /= 2;
28     }
29     return ret;
30 }
31
32 // calculate gcd(a, b)
33 ll gcd(ll a, ll b) {
34     return b == 0 ? a : gcd(b, a % b);
35 }
36
37 // find a pair (c, d) s.t. ac + bd = gcd(a, b)
38 pair<ll, ll> extended_gcd(ll a, ll b) {
39     if (b == 0) return { 1, 0 };
40     auto t = extended_gcd(b, a % b);
41     return { t.second, t.first - t.second * (a / b) };
42 }
43
```

```
44 // find x in [0,m) s.t. ax ≡ gcd(a, m) (mod m)
45 ll modinverse(ll a, ll m) {
46     return (extended_gcd(a, m).first % m + m) % m;
47 }
48
49 // calculate modular inverse for 1 ~ n
50 void calc_range_modinv(int n, int mod, int ret[]) {
51     ret[1] = 1;
52     for (int i = 2; i <= n; ++i)
53         ret[i] = (ll)(mod - mod/i) * ret[mod%i] % mod;
54 }
```

## 2.2 sieve methods : prime, divisor, euler phi

```
1 // find prime numbers in 1 ~ n
2 // ret[x] = false -> x is prime
3 // O(n*loglogn)
4 void sieve(int n, bool ret[]) {
5     for (int i = 2; i * i <= n; ++i)
6         if (!ret[i])
7             for (int j = i * i; j <= n; j += i)
8                 ret[j] = true;
9 }
10
11 // calculate number of divisors for 1 ~ n
12 // when you need to calculate sum, change += 1 to += i
13 // O(n*logn)
14 void num_of_divisors(int n, int ret[]) {
15     for (int i = 1; i <= n; ++i)
16         for (int j = i; j <= n; j += i)
17             ret[j] += 1;
18 }
19
20 // calculate euler totient function for 1 ~ n
21 // phi(n) = number of x s.t. 0 < x < n && gcd(n, x) = 1
22 // O(n*loglogn)
23 void euler_phi(int n, int ret[]) {
24     for (int i = 1; i <= n; ++i) ret[i] = i;
25     for (int i = 2; i <= n; ++i)
26         if (ret[i] == i)
27             for (int j = i; j <= n; j += i)
28                 ret[j] -= ret[j] / i;
29 }
```

## 2.3 chinese remainder theorem

# 3 Data Structure

## 3.1 fenwick tree